1991-12

Specification and analysis of a data transfer protocol using systems of communicating machines

Lundy, Gilbert M.; Miller, Raymond E.

Springer

Lundy, G.M., Miller, R.E. Specification and analysis of a data transfer protocol using systems of communicating machines. Distrib Comput 5, 145157 (1991). https://doi.org/10.1007/BF02252
http://hdl.handle.net/10945/64257

Distrib Comput (1991) 5:145–157

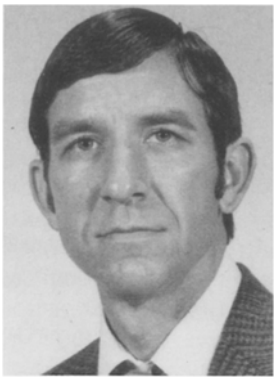
DISTRIBUTED COMPUTING

Specification and analysis of a data transfer protocol using systems of communicating machines

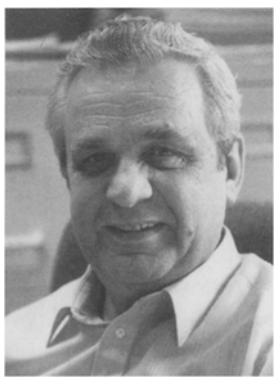
G.M. Lundy[1,*] **and Raymond E. Miller**

[1] Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943, USA
[2] Department of Computer Science, University of Maryland, College Park, MD 20742, USA and NASA Center of Excellence in Space Data and Information Sciences, Goddard Space Flight Center, Greenbelt, MD 20771, USA

Received December 1988 / Accepted July 1991



Gilbert M. Lundy, Jr was born in New Orleans, Louisiana, in 1954. After completing schools in Plano, Texas, he attended Texas A & M University, receiving the B.A. in mathematics (1976). From 1977–81 he served as a Lieutenant in the U.S. Army, based at Fort Ord, California. From 1981–84 he was a software engineer at E-Systems, in Dallas, Texas. During this period he also completed the M.S. program in Computer Science at the University of Texas at Dallas. From 1984 to 1988, he was a graduate student at Georgia Institute of Technology, receiving the Ph.D. in 1988. His research was in the formal modeling of communication protocols for computer networks. Since September 1988, he has been an Assistant Professor of computer science at the U.S. Naval Postgraduate School in Monterey, CA. He teaches classes and performs research in computer networks and communications.



Raymond E. Miller received his Ph.D. degree from the University of Illinois, Urbana-Champaign, in 1957. He was a Research Staff Member at IBM Thomas J. Watson Research Center, Yorktown, Heights, NY, from 1957 to 1980, Director of the School of Information and Computer Science at Georgia Tech from 1980 to 1987, and is currently a Professor of Computer Science at the University of Maryland, College Park, and Director of the NASA Center of Excellence in Space Data and Information Sciences at Goddard Space Flight Center. He has written over 90 technical papers in areas of theory of computation, machine organization, parallel computation, and communication protocols. Dr. Miller is a Fellow of the American Association for the Advancement of Science, a Fellow of the IEEE and a member of ACM. Among his numerous society activities he served as an ACM Council Member-at-Large from 1976–1982, Editor in Chief of the Journal of the ACM from 1972–1976, a Board Member of the Computing Research Association from 1983–1991, and President of the Computing Sciences Accreditation Board from 1985–1987. Currently he is a member of the Board of Governors of the IEEE Computer Society and Vice President for Educational Activities.

Summary. A model for communication protocols called *systems of communicating machines* is used to specify a data transfer protocol with variable window size (e.g., HDLC), which is an arbitrary nonnegative integer, and to analyze it for freedom from deadlocks. The model uses a combination of finite state machines and variables. This allows the size of the specification (i.e., number of states and variables) to be linear in the window size, a considerable reduction from the pure finite state machine model. A new type of analysis is demonstrated which we call *system state analysis*. This is similar to the *reachability analysis* used in the pure finite state model, but it provides substantial simplication by reducing the number of states generated. For example, with the protocol in this paper, if w is the window size, then the global analysis produces $O(w^5)$ states, while the system state analysis produces $O(w^3)$ states. The system state analysis is then combined with an inductive proof, extending the analysis to all nonnegative integers w.

Key words: Communication protocol – Formal model – Specification – Analysis – Formal description technique

1 Introduction

The problem of the specification and analysis of communication protocols has been the subject of much research in recent years. The inherent complexity and variety in

* This research was performed while the authors were at Georgia Institute of Technology

Offprint requests to: G.M. Lundy

these protocols makes this problem a difficult one. Yet protocol correctness is critical to reliable network operation, and standardization and interoperability of protocols is important for ease of network usage. For these reasons the formal modeling and analysis of protocols is an important area of study. In [37], the importance of formal modeling of protocols is discussed. Reference [41] has a comparison of the various techniques for specifying protocols, and [19] discusses the evaluation of formal description techniques. Most of the methods used in modeling protocols can be put into one of the following general classifications: communicating finite state machines, Petri nets, programming languages, and hybrids. A considerable amount of work has been done in each of these areas, and each seems to have advantages and disadvantages both generally and with respect to particular protocols.

1.1 Communicating finite state machines

In the *communicating finite state machine* (CFSM) model, each process is modeled as a finite state machine, and implicit queues between the machines are used for communication. A global state of the network is a tuple containing the state of each machine and the contents of each queue in the system. The most common method of analysis used with this model is called *reachability analysis*. In this method, all possible global states are generated from the initial global state by taking all possible transitions out of each machine. It is well known that if the implicit queues are allowed to have unbounded length, then it is undecidable whether the analysis will terminate, but if an upper bound is placed on the queue length, the method will eventually terminate (see [42]). This model has the advantage of simplicity and a method of analysis that can be easily automated. The obvious disadvantage is that the analysis might not terminate if the queue length is unbounded. A second disadvantage, related to but distinct from the first, is that the number of global states in the reachability analysis is often, for nontrivial protocols, so large as to make the analysis impractical, even if the queue length is bounded, and even if the analysis is automated. This problem is discussed in [22], which contains the automated analysis of a small protocol using this model.

One might argue that the first disadvantage, though of theoretical interest, is not really a problem for practical protocols, because real queues are always of finite length, so we can analyze the protocol of interest using the maximum allowable queue length. This argument has some merit, though the physical bound on queue length is often arbitrary. However, it is our view that the second disadvantage listed is the real problem for this model. The number of global states in the complex protocols used in computer networks grows so quickly that, while a computer analysis might eventually terminate, "eventually" might mean after days or even months of CPU time – clearly impractical.

A third disadvantage is that, with no memory other than the use of states, the specification of a practical protocol can be so complex, containing hundreds of states and transitions, that one can never really be sure it is the intended specification, or grasp an intuitive feeling for what the protocol is intended to do. This problem also manifested itself in the work leading up to [22]. The author was forced to break a 2 machine system into a 6 machine system, because of the difficulty in specifying the protocol with this model.

Much of the work which has been done using this model is an attempt to lessen the effect of these disadvantages, or to get around them. Choi and Miller [8] have worked on simplifying protocols through decomposition. Gouda has done a considerable amount of work in this area. For example, in [11], with Yu, the second problem listed above is addressed. A bounded queue length is assumed, and a method of reachability analysis is used which generates a smaller number of global states then the usual method. Vuong and Cowan also have obtained some results with this model, which address the first problem (decidability). In [42], a class of protocols is defined for which some of the properties are decidable.

1.2 Other models for protocols and parallel programs

Programming languages have the advantage of being more powerful than pure finite state machine models, at the expense of added complexity. The complexity generally makes analysis more difficult. Several papers have appeared in the literature using programming language models, and combinations of these with FSMs, for protocol description. Some of the languages used are CSP, LOTOS, and Ada. CSP (Communicating Sequential Processes) [12, 34] is a high level language with facilities for describing concurrent processes. Several papers on LOTOS have appeared; [4] is a tutorial. Ada [6] contains tasking for parallel or concurrently executing processes, with the rendezvous mechanism for communication between processes. In [40], a data transfer protocol is specified using the language Pascal. That protocol is a host-host, or transport layer protocol. It is verified by proving assertions stating certain desirable properties of the protocol.

There have also been several models which combined programming languages with the finite state model. Estelle is one such approach [5, 9, 20]. In [3], Bochmann and Gecsei describe an alternating bit protocol using a model combining finite state machines and variables. In [39], Shankar verifies a data transfer protocol with variable flow control, using a set of state variables to specify the processes. The protocol allows the channels to make various errors, and *history variables* are defined which are used in the verification. There are several differences between these and our approach. For instance, in Estelle, the machines are specified using an extended form of Pascal.

In [1, 2] a model is presented which had goals very similar to our goals in this work. Protocols were described using a model called "selection/resolution", and a software package was built which served as a tool in helping to evaluate the protocol. That model was also

a precisely defined mathematical model, leading to a possible automated analysis. It was also based on a "state-machine" approach in the description of processes. This was different from our model in several ways, such as in the use of local variables, variables for communication channels, and in the way analysis is carried out.

In [17], Keller presented a model for parallel programs which presented a parallel program as a bipartite, directed graph, with the two node sets representing the states of the program (instructions) and the actions (executions of the instructions). It was similar to this model in the graphical representation of a process, and in allowing (in an extension to the basic model) each process to have local variables. That model, however, was intended for multiple processes executing the same program (as in an operating system), and there are several important differences from the SCM model of this paper. For instance each process executes the same program, simultaneous transitions are not provided for, and no provision is made for the modeling of communication channels.

1.3 Combining finite state machines and variables

The model used in this paper is an attempt to retain the advantages of the pure finite state model, and to reduce or eliminate its disadvantages. To reduce the number of states in each machine, local variables are added. Instead of implicit queues, shared variables are used for communication between processes, and a channel may be modeled as a process explicitly, whenever appropriate. The goal is to stay close to the FSM model, adding only what is needed to reduce its problems, but keeping as much of its simplicity as possible. In particular, a rigorously defined model which can be formally analyzed is essential. So we have started with the FSM model, and taken a step in the direction of the programming language model, but it is a carefully defined, restricted step. The result is a model in which we can do a type of reachability analysis similar to that in the FSM model, but with a reduction in states, both in the specification and in the analysis.

In this paper the model, called *systems of communicating machines*, is used to specify a 1-way data transfer protocol, with a variable window size, which is an arbitrary nonnegative integer. This model, which was previously defined in [30], uses finite state machines and variables. In [3], a similar protocol model is used which has state machines and variables, however, this differs from our model in several ways including the manner in which cannels are represented. In [3], communication between machines is specified as *distantly initiated actions*, and the channels are not explicitly specified. In our model, the channels are explicitly specified as machines, shared variables, or both.

Since the specification is for an arbitrary window size w, the analysis must be also. It turns out that the system state analysis can be defined as a 3-dimensional graph in terms of w. The system state analysis for $w=1$ can then be used as a basis for an induction proof that this graph indeed provides the analysis for all w. So system analysis has been combined with induction to provide a more general analysis than was possible with the pure FSM model.

The contributions of this paper can be now be summarized. First, it demonstrates that the model can be used to specify practical protocols in a way that is clear and concise, easily understandable by implementors, and which also can be analyzed. This is a generalization and extension of [22].

Secondly, as noted above, the specification is general, having an arbitrary window size, and is also of a very reasonable size (linear in the window size), having only $w+1$ states in each process. Such a general specification would not be practical using the pure FSM models, for example, and would certainly have more than a linear number of states.

As for this second point, it may be argued that this protocol can be specified using a programming language with a size that is constant, independent of the window size, so the linear sized specification is not really an advantage. In fact, this protocol can also be specified in this model with a constant number of states; the tradeoff between states and variables is discussed in [21], where 2 different specifications of the same protocol are given. However such an argument misses the point; such a specification does away with the meaning of states, and loss the advantages of the FSM model which we are attempting to retain.

The third point is that the system state analysis has reduced the complexity in analysis from $O(w^5)$ to $O(w^3)$ over the global analysis (Table 1 shows an exact numerical comparison for $0 \leq w \leq 8$), and has been further extended by induction. So there is no need to analyze the protocol for, say, a window of 8, and for a window of 128; the analysis given here applies to both, or to any nonnegative integer. The analysis itself is also worth examining. The system state reachability graph not only has a manageable number of states but it has an interesting 3-dimensional structure, which can be easily generated automatically for use in further analysis, if desired.

Finally, these points and the details of the specification and analysis provide examples suggesting ways in which the model *systems of communicating machines* might be applied to other communication protocols, networks, and parallel systems.

Table 1. Numerical comparison of system and global analysis ($f(w)$ denotes system analysis, $g(w)$ denotes global analysis)

w	$f(w)$	$g(w)$
0	1	1
1	4	8
2	10	60
3	20	240
4	35	700
5	56	1680
6	84	3528
7	120	6720
8	165	11880

Since the *system state analysis* of this model provides a reduced state space analysis, it may be thought of as a projection of the global analysis. Lam and Shankar [16] have verified protocols using a method of projections. In their method, the original protocol was specified using a state transition model – such as the CFSM model – and then a reduced protocol was specified by combining groups of states into a single state, and by grouping messages together. This reduction was generally based upon some particular function of interest, such as connection management or a one-way data transfer. The reduced protocol was then analyzed. In *system state analysis*, the protocol is specified using a specific model – that of this paper – and no alteration is made in the specification itself. Rather, the analysis which is produced is a reduction of the complete global analysis.

In the next section the model is described. Then in Sect. 3, the specification of the protocol is given. Section 4 describes the analysis, called *system state reachability analysis*. In Sect. 5, several other applications of the model are discussed and Sect. 6 concludes the paper.

2 Systems of communicating machines

In this section the model used to specify and analyze the protocol is defined. A more detailed description appears in [21] and [32].

A *system of communicating machines* is an ordered pair $\mathscr{C}=(M, V)$, where

$$M=\{m_1, m_2, \ldots, m_n\}$$

is a finite set of *machines*, and

$$V=\{v_1, v_2, \ldots, v_k\}$$

is a finite set of *shared variables*, with two designated subsets R_i and W_i specified for each machine m_i. The subset R_i of V is called the set of *read access variables* for machine m_i, and the subset W_i the set of *write access variables* for m_i.

Each machine $m_i \in M$ is defined by a tuple $(S_i, s, L_i, N_i, \tau_i)$, where

(1) S_i is a finite set of states;

(2) $s \in S_i$ is a designated state called the *initial state* of m_i;

(3) L_i is a finite set of *local variables*;

(4) N_i is a finite set of names, each of which is associated with a unique pair (p, a), where p is a predicate on the variables of $L_i \cup R_i$, and a is an *action* on the variables of $L_i \cup R_i \cup W_i$. Specifically, an action is a partial function

$$a\colon\ L_i \times R_i \longrightarrow L_i \times W_i$$

from the values contained in the local variables and read access variables to the values of the local variables and write access variables.

(5) τ_i: $S_i \times N_i \longrightarrow S_i$ is a transition function, which is a partial function from the states and names of m_i to the states of m_i.

Machines model the entities, which in a protocol system are processes and channels. The shared variables are the means of communication between the machines. Intuitively, R_i and W_i are the subsets of V to which m_i has read and write access, respectively. A machine is allowed to make a transition from one state to another when the predicate associated with the name for that transition is true. Upon taking the transition, the action associated with that name is executed. The action changes the values of local and/or shared variables, thus allowing other predicates to become true.

Let $\tau(s_1, n)=s_2$ be a transition which is defined on machine m_i. Transition τ is *enabled* if the enabling predicate p, associated with name n, is true. Transition τ may be executed whenever m_i is in state s_1 and the predicate p is true (enabled). The *execution* of τ is an atomic action, in which both the stage change and the action a associated with n occur simultaneously.

The sets of local and shared variables specify a name and a range for each. In most cases, the range will be a finite or countable set of values. For proper operation, the initial values of some or all of the variables should be specified.

A *system state tuple* is a tuple of all machine states. That is, if (M, V) is a system of n communicating machines, and s_i, for $1 \leq i \leq n$, is the state of machine m_i, then the n-tuple $(s_1, s_2, \ldots, s_n)$ is the system state tuple of (M, V). A *system state* is a system state tuple, plus some designation of which outgoing transitions are enabled. That is, two system states are *equivalent* if every machine is in the same state, *and* the same outgoing transitions are enabled. The *initial system state* is the system state such that every machine is in its initial state, and the outgoing transitions are the same as in the initial global state.

The *global state* of a system consists of the system state tuple, plus the values of all variables, both local and shared. It may be written as a larger tuple, combining the system state with the values of the variables. The *initial global state* is the initial system state, with the additional requirement that all variables have their initial values. A global state *corresponds* to a system state if every machine is in the same state, and the same outgoing transitions enabled in the global state are exactly those enabled in the system rate. That is, a global state consists of a tuple of machine states, plus the values of all variables. A system state with the same tuple of machine states and the same enabled outgoing transitions is the corresponding system state.

Note that if the values of all variables are restricted to some finite range, then the model can theoretically be reduced to a simple finite state machine. Otherwise, an infinite number of global states are possible. However, even if the number of global states is infinite, the number of system states is finite, because of the finiteness of each machine. This may allow a reachability analysis on the system states, when a reachability analysis on the global states is infinite. Even when the values of all variables are of a finite range, the number of global states in the equivalent FSM system may be so large as to be intractable. In this paper, it is shown how this model can reduce these difficulties for a specific class of protocols.

The process of generating the set of all system states reachable from the intial system state is called *system state analysis.* This analysis constructs a graph, whose nodes are the reachable system states, and whose arcs indicate the transitions leading from each system state to another. This graph may be generated by a mechanical procedure which consists of the following three steps, repeated as described:

1. Set each machine to its intial state, and all variables to their initial values. The intial set of reachable system states consists of only the initial system state; the initial graph is a single node representing this state.
2. From the current system state vector and variable values, determine which transitions are enabled. For each of these transitions, determine the system state which results from its execution. *If* this state (with the same enabled transitions) has already been generated, then draw an arc from the current state to it, labeling the arc with the transition name. *Otherwise*, add the new system state to the graph, draw an arc from the current state to it, and label the arc with the name of the transition.
3. For each new state generated in step 2, repeat step 2. Continue until step 2 has been repeated for each system state thus generated, and no more new states are generated.

An important question concerned with the practical use of this model is whether the analysis can be automated; that is, whether a program can be written which will produce all of the reachable system states beginning from the initial system state. Such programs have been implemented for the simpler CFSM model [22]. A project to implement the above analysis procedure is currently underway [27]. Due to the use of variables, both local and shared, and the used of enabling predicates which determine whether transitions can be made, the program will be more complex than the automation of the CFSM model. Unlike the CFSM global analysis (it is undecidable whether the CFSM global analysis will ever terminate), *system state analysis* will eventually terminate; there is only a finite number of possible system states. However, due to the complexity – both of the model and of the protocols – our intent is that the program will run under the control of the protocol designer, providing him with the ability to generate portions of the reachability graph, correcting errors in the specification as found, rather than simply generating the entire set of system states, which can be quite large. Such a tool is expected to be quite useful for both the analysis and design of protocols.

A related question is concerned with the insight needed by the designer to apply system state analysis. The procedure is a fairly mechanical one, and can be executed against any protocol specified using the model. Thus one does not need a great deal of understanding of the protocol in order to carry out the analysis. In fact, the process of carrying out the analysis should help the designer or reviewer to gain a greater understanding of the protocol. The program for carrying out this analysis, when completed, should also be a helpful tool in understanding the protocol, as well as in detecting errors. Some insight will be helpful, however, in creating the initial protocol specification. It is desirable to have a balance between states and variables, in order to make the analysis as simple as possible.

3 Specification of a variable window protocol

The protocol which we have chosen to model is a 1-way data transfer protocol with a variable window size, which is essentially a subset of the HDLC (High-level Data Link Control) class of protocols. There are two machines in the system, a sender (m_1) and a receiver (m_2). The sender sends data blocks to the receiver, which are numbered sequentially, $0, 1, \ldots, w, 0, 1, \ldots$ for a window size of w. As in HDLC, the maximum number of data blocks which can be sent without receiving an acknowledgment is w, the window size. The receiver, m_2, receives the data blocks and acknowledges them by sending the sequence number of the next block expected (which is stored in local variable *exp*). The shared variables DATA and SEQ are used to pass messages from sender to receiver, and the shared variable ACK is used to pass acknowledgments back to the sender. The receiver may acknowledge any number of blocks received up to the window size. Upon receiving the acknowledgment, the sender must be able to deduce how many data blocks are being acknowledged. This is done by observing the difference between the values of the received acknowledgment and the sequence number of the last data block sent.

For example, suppose that the window size, w, is 7, and the sender has transmitted 5 data blocks numbered $D0$, $D1$, $D2$, $D3$, $D4$. (We use Dn and An to denote data blocks numered n and acknowledgments numbered n, respectively.) Suppose that the receiver has accepted the first three of these, and chooses to acknowledge them at once. It does so by sending an $A3$ to the sender (as 3 is the number of the next data block the receiver expects). The sender deduces from this that the first 3 data blocks sent have been received. If, however, the sender receives an $A5$, then all 5 data blocks are acknowledged.

The general specification of the protocol is given in Fig. 1 and in Table 2. The specification for a window size $w=1$ is shown in Fig. 2. The state machine diagrams and the variables are shown in Fig. 1, while the action table, containing the enabling predicate and action for each transition is shown in Table 2. Initially, both sender and receiver are in state 0, arrays DATA and SEQ are empty, and ACK is empty. The domains of DATA, *Rdata* and *Sdata* are not specified; these are used to hold user data blocks. *Sdata* and *Rdata* are the interface or access points of the higher layer (user) protocol. The local variables for the sender are *Sdata*, used to store data blocks, *seq*, used to store the sequence number of the next data block to be sent out, and *i*, used as an index into the DATA and SEQ arrays. Initially *seq* is set to 0, and *i* is set to 1. The local variables of the receiver are *Rdata*, *exp*, and *j*. *Rdata* is used to receive and store incoming data blocks, *exp* to hold the expected sequence number of the next incoming data block, and *j* is an index into the shared arrays DATA and SEQ.

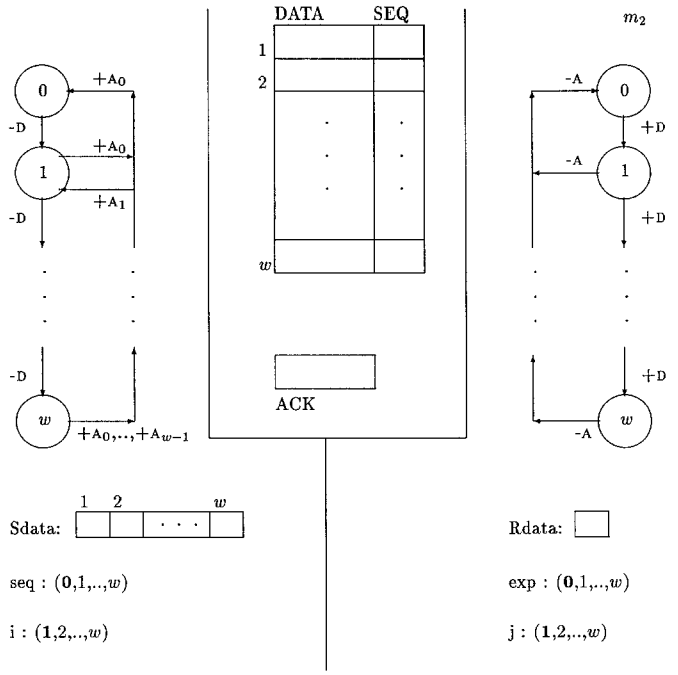


Fig. 1. State machines and variables for the data transfer protocol (initial values are either bold type or empty)

Table 2. Action table for the data transfer protocol (k represents the difference in window size between sender and receiver)

transition	enabling predicate	action
$-D$	DATA(i) $=\mathscr{E} \wedge$ SEQ(i) $=\mathscr{E}$	DATA(i)$\leftarrow$—$Sdata(i)$ SEQ(i)$\leftarrow$—seq $inc(i, seq)$
$+A_k$ $(0 \le k < w)$	ACK $\oplus k$ $= seq \wedge$ ACK $\neq \mathscr{E}$ (next state: k)	ACK$\leftarrow$—$\mathscr{E}$
$+D$	DATA(j) $\neq \mathscr{E} \wedge$ SEQ(j) $= exp$	$Rdata\leftarrow$—DATA(j) DATA(j), SEQ(j)$\leftarrow$—$\mathscr{E}$ $inc(j, exp)$
$-A$	DATA(j) $=\mathscr{E}$	ACK$\leftarrow$—exp

The states of both sender and receiver are numbered 0, 1, ..., w, and each state has an easily recognized intuitive meaning. If the sender is in state 0, then all data blocks sent to date have been received by the receiver, so a full window size of w data blocks may be sent without waiting for an acknowledgment. If m_1 is in state w, then a full window of blocks have been sent, so the sender can only wait for the acknowledgment from the receiver.

If the receiver, m_2, is in state 0, then all received data blocks have been acknowledged. If in state w, then a full window of data blocks have been received, but not acknowledged. Whenever the receiver sends an acknowledgment, all data blocks received up to that point are acknowledged.

The enabling predicate and action for each transition are shown in Table 2. The label or transition name is in the leftmost column, the enabling predicate in the middle, and the corresponding action on the right. There are four basic types of transitions. In the sender, m_1, the $-D$ transition transmits a data block by placing it into the shared variable DATA(i), and the sequence number into SEQ(i). The send is enabled whenever those variables are empty. (The interaction between the sender and the user, or higher layer, is implicit, and not specified here). The *inc* operation increments its arguments, if less than their maximum value, in which case it resets them to the minimum value. The operator $\oplus$ represents the *inc* operation repeated k times, if the argument is k and the symbol $\mathscr{E}$ denotes the empty value. The receive transition in the receiver, m_2, is enabled whenever a data block of the appropriate sequence number is in the jth element of DATA and SEQ. An acknowledgment may be sent by m_2 in any state except 0, in which case no unacknowledged data blocks have been received.

The remaining transition is the $+A_k$, receive acknowledgment, in m_1. If m_1 is in state u, $1 \le u \le w$, and there is a nonempty value in shared variable ACK, then exactly one of the transitions $+A_0$, $+A_1$, ..., $+A_{w-1}$ will be enabled; it will be that A_k such that the predicate ACK $\oplus k = seq$ is true, and the next state is k. In the state diagram, all of the transitions $+A_k$ are shown using the same vertical line for neatness.

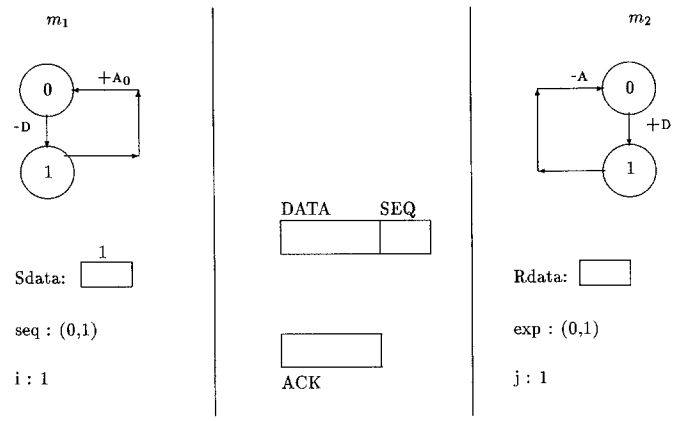


Fig. 2. Specification for $w=1$

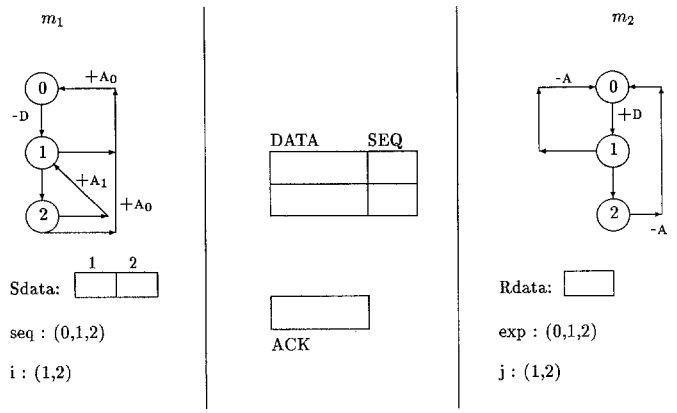


Fig. 3. Specification for $w=2$

The specification for a window size of $w=2$ is shown in Fig. 3. In comparing this with Fig. 2, and referring to the general specification in Fig. 1, one can see how the specification changes as w increases. When w increases by one, the number of states in both sender and receiver increase by one, the range of local variables *seq* and *exp* each increases by one, and the number of elements in the DATA and SEQ arrays increases by one, also. It is shown in the next section that this pattern, which makes the general specification possible, also makes a general analysis possible.

4 Analysis: system state analysis

In this section the system state analysis of the data transfer protocol is given. System state analysis is similar to the reachability analysis used with the pure finite state machine model, but the total number of states which must be generated with system state analysis is significantly smaller. This is explained in more detail in references [31, 32]. First the system state analysis for the window sizes of $w=1$ and $w=2$ are given. From these, it appears that the system state reachability graph is a 3-dimensional geometric structure in the shape of a tetrahedron, and we call this structure a *DT1(w) graph.* Induction is then combined with system state analysis to show that this graph is the system state analysis for a window size w, and from the symmetry of the graph, we can conclude that the protocol is free from deadlocks.

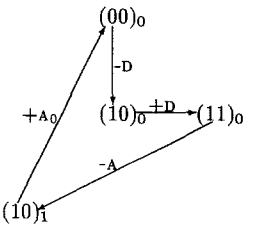


Fig. 4. System state analysis for $w=1$

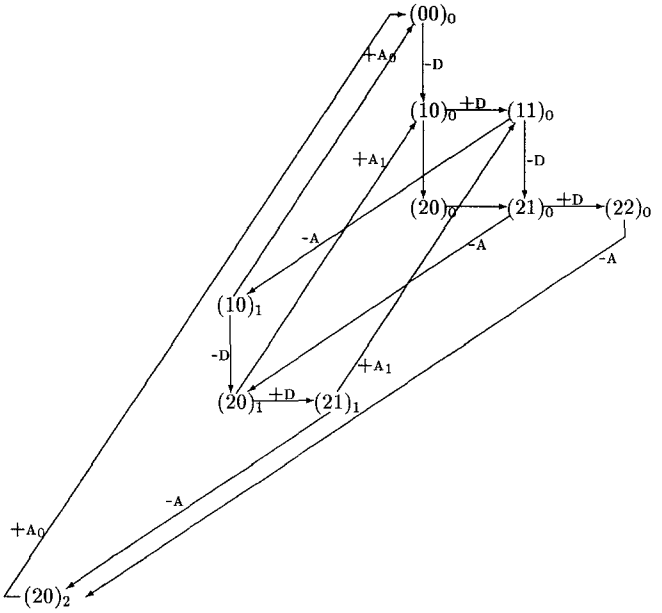


Fig. 5. System state analysis for $w=2$

4.1 Analysis for $w=1, 2$

The specification for $w=1$ was given in Fig. 2, and the system state analysis is given in Fig. 4. Initially both machines are in state 0, the local variables *seq* and *exp* are set to 0 and $i=j=1$. The subscripts are used so that distinct system states having the same tuple (but not the same outgoing transitions) may easily be distinguished. The convention used is that the subscript is initially 0, and is increased whenever a "$-A$" transition is taken, by the number of messages which are being acknowledged. The logic behind this convention will become clear in the next section.

The reader may easily verify that the analysis is as shown in Fig. 4, and should also observe that upon entering the initial state $\langle 0, 0\rangle$ for the second time, that the values of *seq* and *exp* are now 1; that is, the intial *global* state has not yet been reached. In order to reach the initial global state (thus completing the global analysis), the analysis must be continued through four more transitions, generating a total of eight global states.

The analysis for $w=2$ is shown in Fig. 5; the specification was given in Fig. 3. The initial states and variable values are the same as for $w=1$, however there are clearly more states in the analysis.

In comparing the analyses for the window sizes of $w=1$ and $w=2$, it is important to note that the smaller graph is a subgraph of the larger; in fact, either can be obtained easily from the other. The graphs also have a symmetric shape; this is particularly noticeable if the subscripts are taken as the third coordinate in a 3-dimensional cartesian coordinate system, with the states of each machine as the first two coordinates. When this is done, the graph is in the shape of a tetrahedron, with edges which are directed and labeled. Each point in the structure has an intuitive interpretation, which is discussed in the next section.

4.2 Graph definition and structure

In this section we define the directed, labeled graph which was seen to be the system state analysis for window sizes of 1 and 2. The definition is in terms of w, the window size. The definition is used in the next sect. to generalize the analysis to an arbitrary w. In the definition below, the x and y coordinates correspond to the states of sender and receiver, and the z coordinate to the subscript in the analysis.

A *DT1(w) graph* for a nonnegative integer w is a labeled, directed graph, defined by the tuple (N, E, L, ϕ), where

$$N=\{(x, y, z)|0\leq z\leq w, z\leq x\leq w, 0\leq y\leq x-z\}$$

is a finite set of nodes, where each node is specified by an ordered triple;

$$L=\{-D, +D, -A, +A_0, +A_1, \ldots, +A_{w-1}\}$$

is a finite set of labels; the set E of edges is a set of ordered pairs $((x_1, y_1, z), (x_2, y_2, z_2))$ of nodes from N, and is the union of the following four sets:

$$E_1 = \{((x, y, z), (x+1, y, z)) | (x, y, z) \in N, x < w\};$$
$$E_2 = \{((x, y, z), (x, y+1, z)) | (x, y, z) \in N, y < x - z\};$$
$$E_3 = \{((x, y, z), (x, 0, y+z)) | (x, y, z) \in N, y = x - z, x > z\};$$
$$E_4 = \{((x, y, z), (x-z, y, 0)) | (x, y, z) \in N, z > 0\};$$

and the mapping $\phi: E \longrightarrow L$ is defined as follows:

$$\forall (x, y, z) \in E_1, \phi(x, y, z) = -D;$$
$$\forall (x, y, z) \in E_2, \phi(x. y, z) = +D;$$
$$\forall (x, y, z) \in E_3, \phi(x, y, z) = -A;$$
$$\forall (x, y, z) \in E_4, \phi(x, y, z) = +A_k, \quad \text{where } k = x - z.$$

Each node of the graph can be thought of as a point in 3-dimensional space, with nonnegative, integral coordinates (x, y, z). If so, then the structure of the graph is a sequence of $w+1$ triangles, one on top of the other, with the largest triangle at the bottom $(z=0)$ level, and the smallest is a single point at the top $(z=w)$ level. The structure of the $DT1(w)$ graph is shown in Fig. 6. The lower triangle consists of three points, and the top of a single point.

For examples of a $DT1(w)$ graph, the reader should refer back to Figs. 4 and 5; the states of m_1 and m_2 are the x and y coordinates, respectively, and the subscript corresponds to the z coordinate. Now the reasoning behind the convention for subscript use should become clear. The x and y coordinates are the machine states, and the z coordinate corresponds to the difference in the window size of the two machines.

The assignment of labels to the edges can be grouped for an easier intuitive understanding. All of the $-D$ and $+D$ edges occur between points at the same z level. Further, there is a one-to-one mapping between the $-D$ and $+D$ edges at each z level, and all $-D$ edges are perpendicular to the $+D$ edges. The $-A$ and $+A$ edges all go between different levels of z; the $-A$ transitions go from a lower z value to a higher z value and the $+A$ transitions always move down to the bottom $(z=0)$ level. This corresponds to the receiving machine's acknowledgment of all received messages. Also, all the $-A$ edges are parallel or coincident, and all the $+A$ edges are parallel or coincident. The highest level $(z=w)$ is always the single point $(w, 0, w)$. The next higher level consists of 3 points. In moving up, each higher level can be obtained from the previous level by deleting 1 row and increasing the value of both x and z by one.

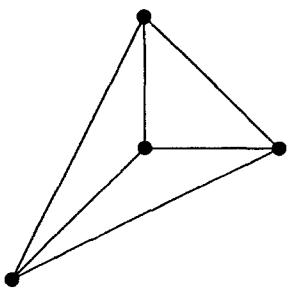


Fig. 6. Structure of the $DT1$ graph

The z coordinate corresponds to the subscript used earlier in the system state analysis. Thus its value corresponds to the number of data blocks which have been acknowledged by m_2, but which have not yet been received by m_1. Put another way, z can be thought of as the difference in window sizes between the two machines.

For example, suppose that beginning in the intial system state $\langle 0, 0 \rangle_0$, m_1 executes two $-D$ transitions, then m_2 receives these two data blocks by executing two $+D$ transitions. The system state at this point would be $\langle 2, 2 \rangle_0$. The z coordinate is 0, as no data blocks have been acknowledged. Next suppose that m_2 executes the $-A$ transition. This transition acknowledges the two data blocks sent by m_1 and received by m_2, and the system state will be $\langle 2, 0 \rangle_2$. The z coordinate is now 2, which represents the number of data blocks which m_2 acknowledged. If m_1 now executes the $+A_0$ transition, the system state will again be $\langle 0, 0 \rangle_0$. The z coordinate is again 0, as all acknowledgments sent by m_2 have been received by m_1.

One of the nice features of the geometric structure of this graph is that the state of the system can be easily inferred from the x, y, z coordinates. For example, at point (5, 3, 0), or system state $\langle 5, 3 \rangle_0$, the sender has transmitted 5 data blocks for which no acknowledgment has yet been received, the receiver has received 3 of these, but acknowledged none. In state $\langle 6, 4 \rangle_2$, the sender transmitted 6 data blocks, but has not yet received an acknowledgment; the receiver received 2 of these, transmitted an acknowledgment of them, and then received 2 more data blocks. The receiver has not yet acknowledged the last 2 data blocks in this state.

4.3 System state analysis for an arbitrary w

In this section the analyses for $w=1$ and $w=2$ are generalized to an arbitrary w, through use of the $DT1(w)$ graph definition. The first 2 lemmas give information about numbers of nodes and edges in a $DT1$ graph. The next 2 lemmas show the relationship between the $DT1$ graphs for various values of w, and between the system state analysis graphs for various window sizes; these are useful in the proof of the theorem. The proofs of the lemmas are given in [21, 30]. The main result is that the analysis for an arbitrary w is given by the $DT1(w)$ graph.

Lemma 1. *The graph* $DT1(w)$ *has* $f(w) = \frac{1}{6}w^3 + w^2 + \frac{11}{6}w + 1$ *nodes.*

Lemma 2. *The graph* $DT1(w)$ *has* $\frac{1}{2}w^3 + 2w^2 + \frac{3}{2}w$ *arcs.*

Lemma 3. $DT1(w)$ *is a subgraph of* $DT1(w+1)$.

Lemma 4. *The system state reachability graph for a window size of w is a subgraph of the system state reachability graph for a window of* $w+1$.

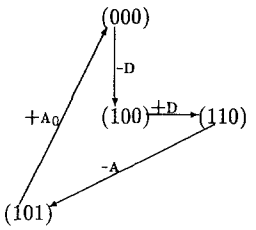


Fig. 7. The graph $DT1(1)$

Theorem 1. *$DT1(w)$ is the system state reachability graph for the 1-way data transfer protocol with a window size of w, where the x and y coordinates are taken as the states of m_1 and m_2, respectively.*

Sketch of proof: The proof is by induction on w, the window size. For $w=0$, the specifications of m_1 and m_2 are a single node with no messages sent or received, so the system state analysis is also the single node $\langle 0, 0\rangle$ and thus the theorem is trivially true. We use the case $w=1$ as a basis for the induction.

The specification for $w=1$ is given in Fig. 2, and the system state analysis is given in Fig. 4. This analysis has 4 system states. The graph $DT1(1)$, shown below, also has 4 nodes: 000, 100, 110, and 101. These correspond to the system states of the analysis, and it is clear that the edges between these are identical to the transitions of the system state analysis.

For the inductive hypothesis, assume that the theorem is true for some positive integer n. That is, assume that $DT1(n)$ is the system state reachability graph for the data transfer protocol with a window size of n. We must show that the theorem holds for $n+1$.

In what follows, we use the notation '$SA(n)$' and '$GA(n)$' to denote the system state analysis and the global state analysis for a window size of n, respectively.

By Lemma 4, $SA(n)$ is a subgraph of $SA(n+1)$, and by the inductive hypothesis $SA(n)$ is the graph $DT1(n)$; so we shall take the specification for $n+1$, and the graph $DT1(n)=SA(n)$ as a partial analysis, and complete the system state analysis of the protocol for $n+1$, giving $SA(n+1)$, showing step by step that this completed analysis of protocol $n+1$ is the graph $DT1(n+1)$.

Consider each node in $SA(n)=DT1(n)$, and add the additional transitions which exist in the protocol $n+1$, which are not in protocol n. The z coordinate takes on the values $0, 1, \ldots, n$. At the $z=0$ level the nodes/states of $SA(n)$ are

000,
100, 110,
...,
...,
$n00, n10, \ldots, nn0$.

For the x values of $0, 1, \ldots, n-1$ (that is, all but the last row as shown above), there are no additional transitions. This is because the transitions possible out of those states are exactly the same in the n protocol as in the $n+1$ protocol; m_1 may only send data blocks, and m_2 may only receive data blocks or send acknowledgments at the $z=0$ level; and for all but the last row, these transitions already exist in $DT1(n)$. But at the last row – states $n00, \ldots, nn0$ – there is an additional transition possible in protocol $n+1$ which is not possible in protocol n: a "$-D$" transition out of m_1. This is because m_1 has one more state in the $n+1$ protocol, and the corresponding variables can take on one additional value.

These "$-D$" transitions out of the states of the bottom row above generate another row of states at this $z=0$ level:

$(n+1, 0, 0), (n+1, 1, 0), \ldots, (n+1, n, 0)$.

For each of these new states, there is one additional transition possible, which is the corresponding "$+D$" in m_2. This is possible because m_2 also has one additional state in the $n+1$ protocol, and the corresponding variable values take on one additional value. These "$+D$" transitions are from state $(n+1, i, 0)$ to state $(n+1, i+1, 0)$ for $0 \le i \le n$. This generates one more state, $(n+1, n+1, 0)$. This gives a total of $n+1$ additional states at the $z=0$ level, as well as n additional "$-D$" and "$+D$" transitions at that level.

There are no "$+A$" transitions possible out of these new states; this is because no data blocks have been received which are not yet acknowledged. However there is a single "$-A$" transition which is possible (in fact, the only transition possible) out of state $(n+1, n+1, 0)$. This is possible because all data blocks sent by m_1 have been received by m_2 in this state. The sender, m_1, is unable to send any more data blocks. So m_2 may take a "$-A$" transition, which leads to the system state $\langle n+1, 0\rangle$. But this is not the same system state as $(n+1, 0, 0)$ already generated; because the only transition possible out of this new state is "$+A_0$" which leads back to the initial system state. (This is easily seen by observing the sequence of transitions which have occurred since the initial state.) So we assign a new z value to this state, $n+1$, giving the node $(n+1, 0, n+1)$. (We choose the value for z to be $n+1$ because this is exactly the number of data blocks which are being acknowledged; this is consistent with the interpretation of the meaning of z for $DT1(n)$.)

From the definition of $DT1(w)$, it is easily seen that the additional nodes as shown above are exactly the additional nodes in $DT1(n+1)$ at the $z=0$ level not in $DT1(n)$, as well as the additional transitions.

To complete the proof, the analysis is completed similarly for the $z=2, 3, \ldots, n$ levels and this is done in [21, 30].

The significance of this theorem is that the analysis for a class of data transfer protocols is completed for an arbitrary window size. Further, because every node in the graph has a path back to the initial state, there can be no deadlock nodes. This shows that the method of system state analysis has the potential to greatly simplify analysis of at least some types of protocols.

By this result, the graph of Fig. 5 is the analysis for the protocol of window size 2. The first 2 digits of each node are the states of machines m_1 and m_2, respectively.

The node 000 is the initial system state. There is an intuitive meaning for each system state, analogous to the meaning of the states in the specification. In 000, all data blocks sent have been received and acknowledged, and the acknowledgment has been received. At the entire $z=0$ level, all acknowledgments that have been sent have been received. At the $z=1$ level, exactly 1 acknowledgment has been sent but not received. The $-A$ transitions, which acknowledge data blocks, go up in their z value by exactly the number of data blocks being acknowledged. For example, note that the $-A$ from state 220 goes to state 202. Since z increases by 2, this is the number of data blocks m_2 is acknowledging. Tracing the paths from the initial state 000 to 220, it is seen that exactly 2 data blocks have been sent and received, as expected.

4.4 *Comparison with global analysis*

As previously stated, the global analysis has $O(w^5)$ reachable states, as compared with $O(w^3)$ reachable states in the system state analysis.

To see this, consider the system state analysis for $w=2$ (Fig. 5); it has

$$\tfrac{1}{6}(2)^3+(2)^2+\tfrac{11}{6}(2)+1=10$$

states. In carrying out the system state analysis, the initial *global* state is

$$(0, 0, seq=0, exp=0, i=1, j=1, \text{ACK}=\mathscr{E}).$$

If the system takes the following progression

$$\langle 0,0\rangle \longrightarrow -D \longrightarrow \langle 1,0\rangle \longrightarrow +D \longrightarrow \langle 1,1\rangle \longrightarrow -A \longrightarrow \langle 1,0\rangle \longrightarrow +A_0 \longrightarrow \langle 0,0\rangle$$

then the global state at the second occurrence of $\langle 0,0\rangle$ is

$$(0, 0, seq=1, exp=1, i=2, j=2, \text{ACK}=\mathscr{E}).$$

This is the same system state as at the first occurrence, but *not* the same global state. It turns out that for $w=2$, there are $60=(10)(2)(3)$ global states. That is, there is a global state for each possible combination of $seq=exp$ and $i=j$ when the 2 machines are in the initial state, and each such global "initial" state produces $f(w)$ global states. The global analysis consists essentially of $w(w+1)$ system state analyses. This result is given in the following theorem. First 2 lemmas are given, which are used in the proof of the theorem.

Lemma 5 states an important fact about the relationships between the variables of m_1, m_2, and the shared variables whenever both machines are in state 0. The final lemma is basic to the applicability of system state analysis.

Lemma 5. *Consider the data transfer protocol of Fig. 1. For any reachable system state* $\langle 0,0\rangle$, *we must have* $i=j$, $seq=exp$, *and* $\text{DATA}(k)=\text{SEQ}(k)=\text{ACK}=\mathscr{E}$ *for* $1\leq k\leq w$.

Lemma 6. *For the data transfer protocol specified by Fig. 1, any reachable global state in which both* m_1 *and* m_2 *are in state 0 corresponds to the initial system state, and yields the* $DT1(w)$ *graph when system state analysis is applied.*

Theorem 2. *The global analysis for the data transfer protocol has*

$$g(w)=w(w+1)f(w)=w(w+1)(\tfrac{1}{6}w^3+w^2+\tfrac{11}{6}w+1)$$

states, where w denotes the window size.

Sketch of proof: By Theorem 1, there are at least $f(w)$ global states, because there must be at least 1 global state corresponding to each of the system states. In order to prove this theorem, it is shown that there are exactly $w(w+1)$ global states corresponding to each of the $f(w)$ system states. The global analysis for $w=1$ is shown in Fig. 8 and Table 3. This provides a proof of that case, so in what follows we may assume that $w\geq 2$.

First we show that for the initial system state, there are $w(w+1)$ corresponding global states.

The initial system and global state is

$$\langle 0, 0, seq=0, exp=0, i=1, j=1, \text{DATA}=\mathscr{E}, \text{SEQ}=\mathscr{E}, \text{ACK}=\mathscr{E}\rangle.$$

If the transitions $-D$, $+D$, $-A$, $+A_0$ are executed (the reader may observe that these transitions are enabled in the resulting sequence of states) then the resulting global state is

$$\langle 0, 0, seq=1, exp=1, i=2, j=2, \text{DATA}=X, \text{SEQ}=0, \text{ACK}=\mathscr{E}\rangle.$$

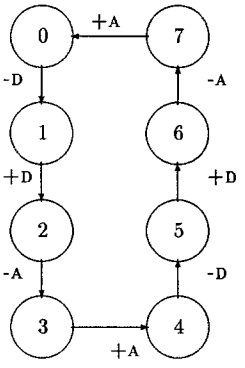


Fig. 8. Global analysis for $w=1$

Table 3. Global states for $w=1$

num	*state*	*seq*	*i*	*exp*	*j*	DATA	SEQ	ACK
0	$\langle 0,0\rangle_1$	0	1	0	1	$\emptyset$	$\emptyset$	$\emptyset$
1	$\langle 1,0\rangle_1$	1	1	0	1	X	0	$\emptyset$
2	$\langle 1,1\rangle_1$	1	1	1	1	$\emptyset$	$\emptyset$	$\emptyset$
3	$\langle 1,0\rangle_2$	1	1	1	1	$\emptyset$	$\emptyset$	1
4	$\langle 0,0\rangle_2$	1	1	1	1	$\emptyset$	$\emptyset$	$\emptyset$
5	$\langle 1,0\rangle_3$	0	1	1	1	X	1	$\emptyset$
6	$\langle 1,1\rangle_2$	0	1	0	1	$\emptyset$	$\emptyset$	$\emptyset$
7	$\langle 1,0\rangle_4$	0	1	0	1	$\emptyset$	$\emptyset$	0

This is the initial system state (the same enabled outgoing transition), but a different global state. Repeatedly executing the sequence of transitions $-D$, $+D$, $-A$, $+A_0$ generates the global states

$\langle 0, 0, seq=2, exp=2, i=3, j=3,$
$\text{DATA}=\mathscr{E}, \text{SEQ}=\mathscr{E}, \text{ACK}=\mathscr{E}\rangle$
$\langle 0, 0, seq=3, exp=3, i=4, j=4,$
$\text{DATA}=\mathscr{E}, \text{SEQ}=\mathscr{E}, \text{ACK}=\mathscr{E}\rangle$
$\langle 0, 0, seq=w-1, exp=w-1, i=w, j=w,$
$\text{DATA}=\mathscr{E}, \text{SEQ}=\mathscr{E}, \text{ACK}=\mathscr{E}\rangle$
$\langle 0, 0, seq=w, exp=w, i=1, j=1,$
$\text{DATA}=\mathscr{E}, \text{SEQ}=\mathscr{E}, \text{ACK}=\mathscr{E}\rangle$
$\langle 0, 0, seq=0, exp=0, i=2, j=2,$
$\text{DATA}=\mathscr{E}, \text{SEQ}=\mathscr{E}, \text{ACK}=\mathscr{E}\rangle$
$\langle 0, 0, seq=0, exp=0, i=1, j=1,$
$\text{DATA}=\mathscr{E}, \text{SEQ}=\mathscr{E}, \text{ACK}=\mathscr{E}\rangle$

There are $w(w+1)$ such global states, 1 for every possible combination of $i=j$ (w values) and $seq=exp$ ($w+1$ values).

Thus there at least $w(w+1)$ global states corresponding to the initial system state, as each of these clearly have the same outgoing transition enabled. If there are any more global states corresponding to the system state $\langle 0, 0\rangle$, then either $i \neq j$, or $seq \neq exp$, or one of DATA, SEQ, and ACK must be nonempty. But from Lemma 5, this cannot occur, so there are exactly $w(w+1)$ global states corresponding to system state $\langle 0, 0\rangle$.

Since each of the $w(w+1)$ global states corresponds to the initial system state, by Lemma 6, system state reachability analysis may be applied to each of these global states to generate $f(w)$ system states. What remains is to show that each global state thus formed is a unique global state and that there are no other global states. This is done by using the uniqueness of the initial global states, and the relationships between the variables. The reader is invited to complete the details; a complete proof is in [21, 30].

5 Other applications of the model

This paper has covered the application of *systems of communicating machines* to a particular type of protocol; but there are numerous other possible applications, some of which are discussed briefly in this sect.

In [28], SCM was used to model the token ring protocol, which is standardized as IEEE 802.5, *Token Ring Access Method* [15]. This is a local area network in which an arbitrary number of stations are connected, each with exactly one input link and one output link, forming a cycle. The token passing and data transfer phases of the protocol were specified. The *system state analysis* of that specification was presented in [36]. The analysis showed that the protocol was free from deadlocks and nonexecutable transitions.

In [29], the CSMA/CD protocol was specified and verified. The analysis showed that the protocol was free from deadlocks, and that, except for the case of continuous collisions, that the protocol has progress properties. This protocol, also called "ethernet," is widely used in university networks across the USA, and is IEEE Standard 802.3 [13]. The analysis was based on a system state analysis for the 2-machine case, which was then extended to the case of an arbitrary number of machines. Thus it is similar to that of this paper, in that we began with a finite system state analysis and extended it; but in this paper we extended a small window size to an arbitrary size, while in the CSMA/CD paper we extended the size of the network from two machines to an arbitrary size. One of the unique aspects of that work was in the use of shared variables to model collisions. These were modeled by simultaneous writes by different machines to a shared variable.

The token bus protocol [14] was specified by SCM in [24], and analyzed in [7]. The analysis used *system state analysis* for the case of a 2-, 3- and 4-machine network. These analyses then were used to show that the properties proved applied to an arbitrary sized network. The protocol was shown to be free from deadlocks and nonexecutable transitions, and to possess progress properties.

A specialized network protocol with a centralized access control was modeled in [26]. This protocol is widely used to connect various signal processing devices and computers on military aircraft. It is a different type of local area network, in which communication is controlled by a central monitor, a computer which controls the electronic devices on the network.

FDDI (Fiber Distributed Data Interface) is a recent standard which specifies a 100 Mbps (Mbps $=10^6$ bits/sec) network using optical fiber. In [25], SCM was used to specify this protocol, and to verify it. In this work, the definition was extended to allow for the explicit timing of transitions. Timing is a critical part of this network, and this work showed that SCM also has the capability of specifying timing information. The verification showed that the network is free from errors such as deadlock, and also discussed the timing properties and how they are satisfied.

The examples, as well as the application of this paper, show that the model *systems of communicating machines* has a wide range of applications to communication networks. It can also be used for more general parallel systems. Research which is currently in progress is using the model SCM as a tool for the specification real time systems [18]. The definition of the model is used as a starting point for the definition of a general "real time system." Then specific cases of real time systems are specified.

Other current research is concerned with formally specifying a protocol which has been suggested as an improved FDDI [10, 23]. This work is a very detailed protocol specification using SCM, and a modification to the FDDI protocol as well.

It is also worth noting that three recent standards documents use a method of specifying the protocol which has definite similarities to the model *system of communi-*

cating machines [43, 15, 35]. The methods used in these documents also have finite state machines and variables; however these are not formally defined models, and this leads to some degree of informality. But the use of such a method in standards documents seems to indicate that a significant part of the standards community accepts the use of finite state machines and variables as a reasonable method for the specification of real-world protocols.

Another possible application of *systems of communicating machines* is as an analysis tool for protocols specified using other models such as the standard FDTs Estelle and LOTOS. Since Estelle has an extended finite state machine as its underlying model, one possible approach would be to transform the Estelle specification into a *systems* specification, then exercise *system state analysis*. Some restrictions on the constructs which the Estelle specification contains may be necessary, however, in order to get a complete translation and analysis. For example, in [38], in order to analyze a specification, some Estelle statements and clauses were eliminated. A LOTOS specification would probably be more difficult, in general, to transform into *systems of communicating machines*; however the effort might possibly provide a more easily analyzable specification.

The diversity of the kinds of protocols that have been modeled by the *systems of communicating machines* formalism provides extensive evidence as to its versatility for formally specifying and verifying the correctness of protocols of different types, yet providing a model that is simple enough to allow verification and other studies of the behavior of the protocol.

6 Conclusions

A model for specifying protocols, called *systems of communicating machines*, was used to specify a 1-way data transfer protocol, and the protocol was analyzed for freedom from deadlocks using a method called *system state analysis*. Both specification and analysis are for an arbitrary window size.

The contributions of this paper are the specification of a widely used, practical protocol using a precisely defined yet easily understood model; the generality of the specification (any window size); and the analysis of the protocol, particularly the reduction in the size of the state space over previous models and in the use of induction combined with the system state analysis. Additionally, the details of the specification and analysis suggest ways in which the model might be applied to other protocols and networks. We have also described the general procedure for carrying out the analysis, and some other applications of the model are also briefly discussed.

The specification of the protocol has a number of states which is linear in the window size. This is a sizeable reduction in the number of states over a pure finite state machine (FSM) representation of the same protocol, which requires a number of states which is at least a quadratic function of the window size. The analysis produces a graph which has a number of nodes and edges which are cubic in the window size. This is a reduction from $O(w^5)$ for the global analysis. Further, in both the specification and the analysis, each state has an easily understood intuitive meaning.

There are several questions which remain, and areas open for further work. It remains to be shown under exactly what conditions system state analysis may be applied in the place of global analysis. In this particular protocol it is clear that the analysis is valid, because we have shown the exact relationship between the 2 analyses, but the result should be proven for a more general case.

Another question is concerned with the proper balance between states and variables in the specification of the protocol. A specification with too many variables and too few states makes the *system state analysis* meaningless; a specification with fewer variables and more states than needed increases the complexity of the analysis. Some insight on the part of the designer will be helpful in making the specification; some trial and error may be necessary. Once the specification has been made, however, the analysis can be carried out mechanically. Each step in the analysis may be more complex than in a pure finite state analysis, however the drastic reduction in the number of states generated makes this increase worthwhile.

The protocol modeled here assumes no errors in transmission and does not make use of timers. In [33], the model is used to specify a similar, full duplex protocol. It is believed that the results of this paper can be combined with a synthesis or decomposition approach; another possibility is that the projection method of [16] might be adapted. Additional consideration has been given to modeling errors and timers. Machines can be used to model channels which explicitly model errors, as well as timers. It is clear that this model and the system state analysis approach will be helpful in analysis which includes these features in the protocol specification.

References

1. Aggarwal S, Barbara D, Meth KZ: SPANNER: a tool for the specification, analysis and evaluation of protocols. IEEE Trans Software Eng SE-13: 1218–1237 (1987)
2. Aggarwal S, Kurshan RP, Sharma D: A language for the specification and analysis of protocols. Protocol specification, testing and verification III. North-Holland 1983
3. Bochmann GV, Gecsei J: A unified method for the specification and verification of protocols. Information Processing, North Holland Publishing Company 1977, pp 229–234
4. Brinksma E: A tutorial on LOTOS. Proc IFIP WG 6.1 5th Int Workshop on Protocol Specification, Testing and Verification. Toulouse-Moissac, France, June 10–13, 1985
5. Budkowski S, Dembinski P: The formal specification technique estelle. Comp Networks ISDN Syst 14 (1987)
6. Castenet R, Dupuex A, Guitton P: Ada, a well-suited language for the specification and implementation of protocols. Proc IFIP WG 6.1 5th Int Workshop on Protocol Specification, Testing and Verification, Toulouse-Moissac, France, June 10–13, 1985
7. Charbonneau LJ: Specification and analysis of the token bus protocol. M.S. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA 1990

8. Choi TY, Miller R: Protocol analysis and synthesis by structured partitions. Comp Networks ISDN Syst 11 (5):367–381 (1986)
9. Diaz M, Ansart JP, Courtiat J, Azema P, Chari V: The formal description technique Estelle. North-Holland Elsivier 1989
10. Elmiro L: Modeling an improved FDDI protocol. M.S. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA (in preparation)
11. Gouda M, Yu YT: Maximal progress state exploration. ACM SIGCOMM Symposium. University of Texas at Austin, March 8–9, 1983
12. Hoare CAR: Communicating sequential processes. CACM 21 August 8, 1978
13. Institute of Electrical and Electronic Engineers. IEEE Standard 802.3: Carrier sense multiple access with collision detection access method and physical layer specification 1985
14. Institute of Electrical and Electronic Engineers. IEEE Standard 802.4: Token-passing bus 1985
15. Institute of Electrical and Electronic Engineers. IEEE Standard 802.5: Token ring access method and physical layer specification 1985
16. Lam SS, Shankar U: Protocol verification via projections. IEEE Trans Software Eng SE-10 (4):474–491 (1984)
17. Keller RM: Formal verification of parallel programs. Commun ACM 371–384 (1976)
18. Kvaslerud O: Applications of high speed networks. M.S. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA 1991
19. Le Moli G: An approach for evaluating formal description techniques. Proc IFIP WG 6.1 5th Int Workshop on Protocol Specification, Testing and Verification. Toulouse-Moissac, France, June 10–13, 1985, North-Holland
20. Linn RJ: The features and facilities of estelle: a formal description technique based upon an extended finite state machine model. Proc IFIP WG 6.1 5th Int Workshop on Protocol Specification, Testing and Verification. Toulouse-Moissac, France, June 10–13, 1985
21. Lundy GM: Systems of communicating machines: a model for communication protocols. Ph.D. Thesis, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 1988
22. Lundy GM: Improving throughput in the FDDI token ring network. In: Johnson M (ed) The second IFIP Int Workshop on Protocols for High Speed Networks, Palo Alto, CA 1990, pp 369–382
23. Lundy GM: Modeling and analysis of data link protocols. TN 86-499.1. Telecommunications Research Laboratory, GRE Laboratories, Inc, 40 Sylvan Road, Waltham, MA January 1986
24. Lundy GM: Specification and analysis of the token bus protocol using systems of communicating machines. IEEE Systems Design and Networks Conference, Santa Clara, CA 1990
25. Lundy GM, Akyildiz IF: A formal model of the FDDI network protocol. In: Europa Proceedings of the EFOC/LAN '91, London 1991, pp 201–205
26. Lundy GM, Christensen P: Specification of the MIL-standard 1553 protocol using systems of communicating machines. IEEE military communications conference, Monterey, CA 1990
27. Lundy GM, Locke J: Automated design and analysis of protocols. Tech Rep, Department of Computer Science, Naval Postgraduate School 1991 (in preparation)
28. Lundy GM, Luqi: Specification of a token ring protocol using systems of communicating machines. IEEE systems design and networks conference, Santa Clara, CA 1989
29. Lundy GM, Miller RE: Analyzing a CSMA/CD protocol through a systems of communicating machines specification (submitted for publication)
30. Lundy GM, Miller RE: Specification and analysis of a general data transfer protocol. Tech Rep GIT-88/12. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 1988
31. Lundy GM, Miller RE: A variable window protocol specification and analysis. Eighth International Symposium on Protocol Specification, Testing and Verification, Atlantic City, NJ, June 7–10, 1988
32. Miller RE, Lundy GM: An approach to modeling communication protocols using finite state machines and shared variables. IEEE Global Telecommunications Conference, Houston, TX, December 1–4, 1986
33. Miller RE, Lundy GM: A model for communication protocols using finite state machines and shared variables. Tech Rep GIT-ICS-86/22, Georgia Institute of Technology, Atlanta GA 30332, October 23, 1986
34. Nikolaou C, Clarke E, Nisson F, Shuman S: A methodology for verifying request processing protocols. ACM SIGCOMM, University of Texas at Austin, March 8–9, 1983
35. Proposed draft standard. Distributed queue dual bus subnetwork of a metropolitan area network, IEEE 802.6
36. Raiche C: Specification and analysis of the token ring protocol. M.S. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA 1989
37. Rudin H: An informal overview of formal protocol specification. IEEE Communications Magazine 23 (3):46–52 (1985)
38. Sarikaya B, Bochmann G, Cerny E: A test design methodology for protocol testing. IEEE trans software eng SE-135 (1988)
39. Shankar Udaya: Verified data transfer protocols with variable flow control. Tech Rep CS-TR-1746, UMIACS-TR-86-25. Department of Computer Science, University of Maryland, College Park, MD, December 1986
40. Stenning NV: A data transfer protocol. Comput Networks 1:99–110 (1976)
41. Venkatramen RC, Piatkowski TF: A formal comparison of formal protocol specification techniques. Proc IFIP WG 6.1 5th Int Workshop on Protocol Specification, Testing, and Verification. Toulouse-Moissac, France, June 10–13, 1985, North-Holland
42. Vuong ST, Cowan DD: Reachability analysis of protocols with FIFO channels. ACM SIGCOMM, University of Texas at Austin, March 8–9, 1983
43. X3T9 committee of ANSI. FDDI token ring media access control. ANSI Standard X3T9.5, 1990